**Class** : User defined blue print from which Objects are creaeted.

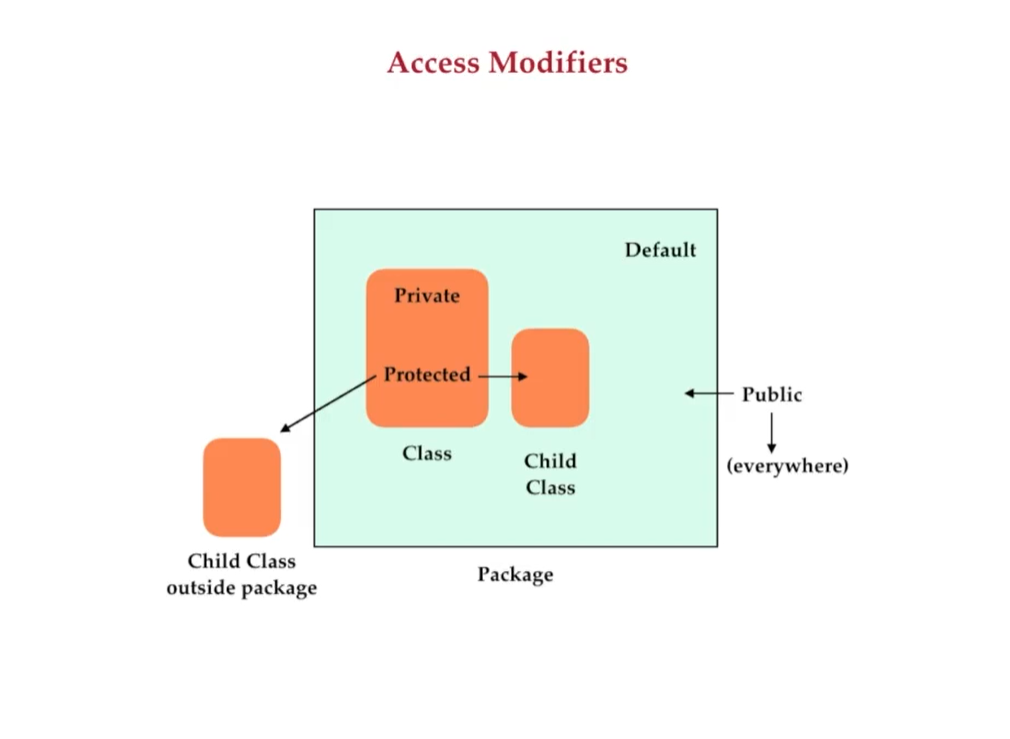
Class Dog -> Object Bichu (type:JRT)

**Variables :**

* **Instance Variables** Variables defined in Class
* **Local Variables** Variables defined in functions
* **Static Variables** Variables defined as static are shared between instances.

**Access Modifiers :**

* **Public –** Can be accessed from everywhere
* **Private –** Can be accessed within the class
* **Protected –** Same packaged and within subclasses
* **Defaiult –** Can be only accessed within the package



**Abstract**

* Used to hide background details of the features.
* Can contain full functions that can be shared between all the subclasses. Though subclasses may chose to ovveride them.
* Cannot be instantiated

**Interfaces**

* Blueprint of a class.
* Can only have method declaration.
* Can have method bodies, through default, private and static declaration of methods.

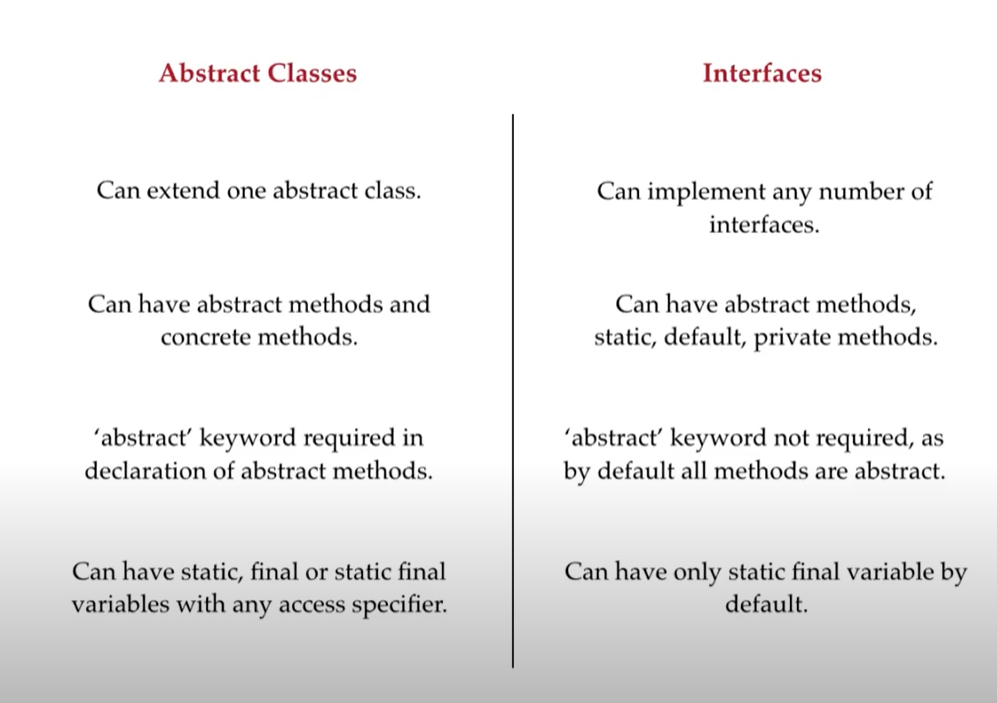
**Final**

* Used to finalize the implementations of class,variables and methods. These items cannot be inherited , changed or overriden.

**Static**

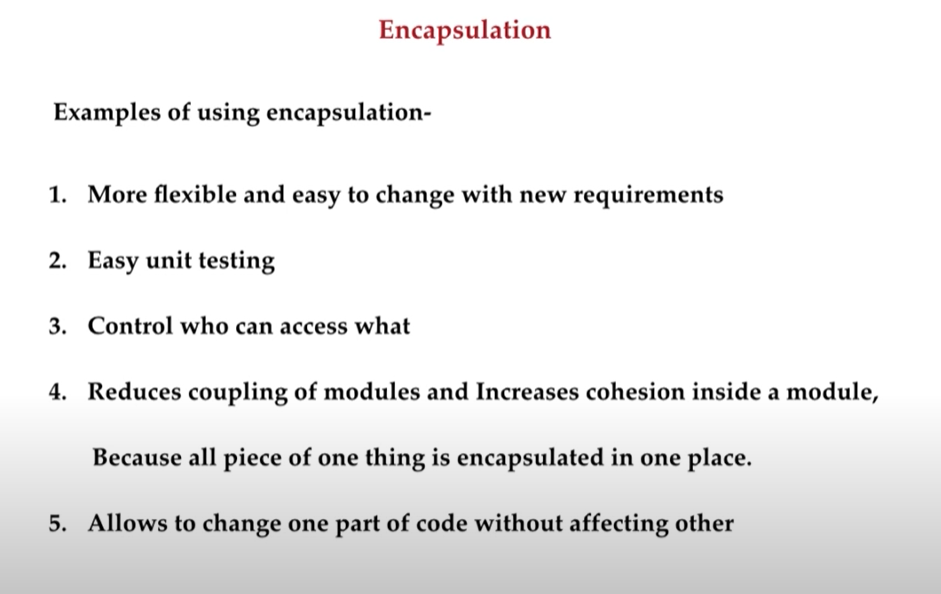
* Used to allocate a memory for variables or methods. These things can be accessed without the need for instances.
* Static variables declared are shared in all the instances.
* Static variables are assigned to memory during class loading.

**Data Hiding –** is the process of hiding internal object details, i.e. data members.



**Encapsulation**

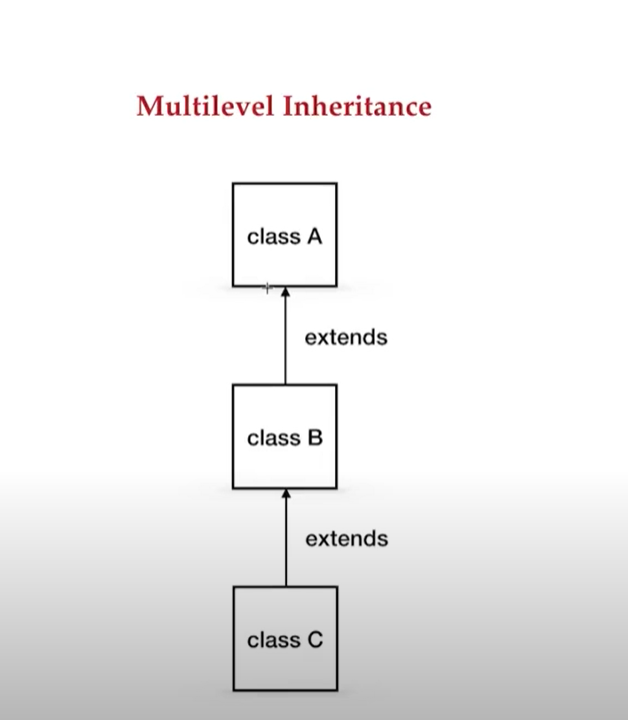
* Grouping of data members and and member functions into a single unit.
* Java classes are examples of encapsulation.
* Encapsulation hides the background information.
* Denies direct access to the variables.



**Inheritance**

* Process of creating new classes by deriving from other parent class.
* IS A and HAS A Relationship possible
* Method Ovveriding is another important aspect

Parent Class -> Child Class, Exception -> IOException



Suppose Employee class has Manager and Developer subclasses. So these classes derive from Employee class.

1. Single Inheritance
2. Multiple Inher itance

1. Multilevel Inheritance -

When subclass are more than 1 parent (Can cause diamond ambiguity problem, same methods in different parent classes when called gets confused whom to invoke.). Java doesnt support Multilevel. Can implement interfaces

1. Hybrid Inheritance

Multiple + Multilevel , Not possible

1. Hierarchical Inheritance

Single Parent more than one children.

1. Cyclic

Parent extends child, child extends parent. Not required and supported in Java. We can create one class.

**Association –** Establishing relationship between two different classes throught their objects.

* Aggregation – If Both entities associated can exist independently. Example – College has teachers.
* Composition – Stronger form of association , the two entities are highly dependent on each other.

**Polymorphishm –** is the ability of something to take many forms.

Function signature = **MethodName (<Argument Type>)**

* **Method Overloading (Compile Time PolyMorphishm since compile keep the method signatures) -**  Functions with the same name but different set of argument types. Its also called early binding.
* **Method Overriding (Runtime polymorphism) –** Type of the referrred object determinds which version of method overriden will be executed.

Base base= new Base() -> base method will be called

Derived der=new Derived() -> derived method will be called

Base obj=new Derived() -> Derived method will be called

Derived obj=new Base() XX Not possible

We won’t be able to decrease the visibility of the derived class. We can increase the visibility of the derived class.

We cannot override static and final methods. Private methods cannot be overriden either.

We can access base class methods using super keyword.

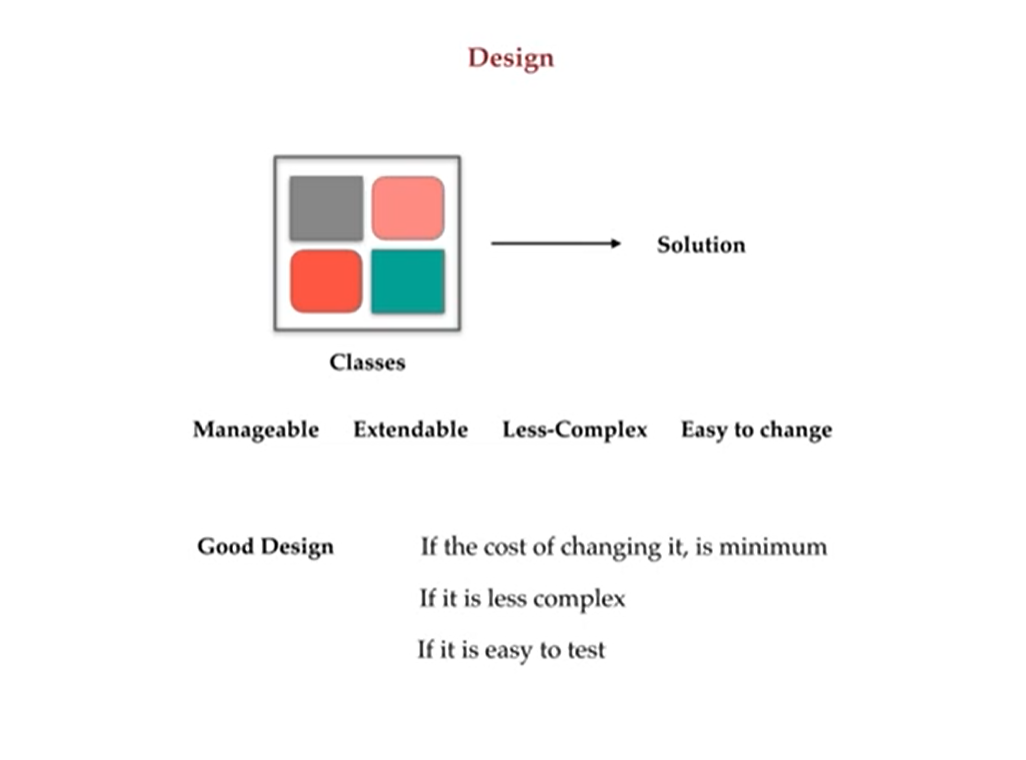
**Abstraction**

Abstraction is hiding unnecessarry details or Exposing that is necessary to consumer classes /methods.

Higher Abstraction leads to simplicity, and it requires trust.

**Cohesion –** Methods will stay where it is appropriate. Loose Coupling should be present.

**Design Principles**



DRY – DONT REPEAT YOURSELF

KISS – KEEP IT SIMPLE AND STUPID

YAGNI – YOU ARENT GONNA NEED IT

SLAP – SINGLE LEVEL OF ABSTRACTION PRINCIPLE

**SOLID** – **SINGLE RESPONSIBILITY PRINCIPLE**

-Each class should have a single responsiblity

**OPEN AND CLOSE PRINCIPLE**

-Classes should be open to extensions and closed to modifications.

**LISKOV SUBSTITUTION**

-Parent and Child class should be interchangeable without breaking anything

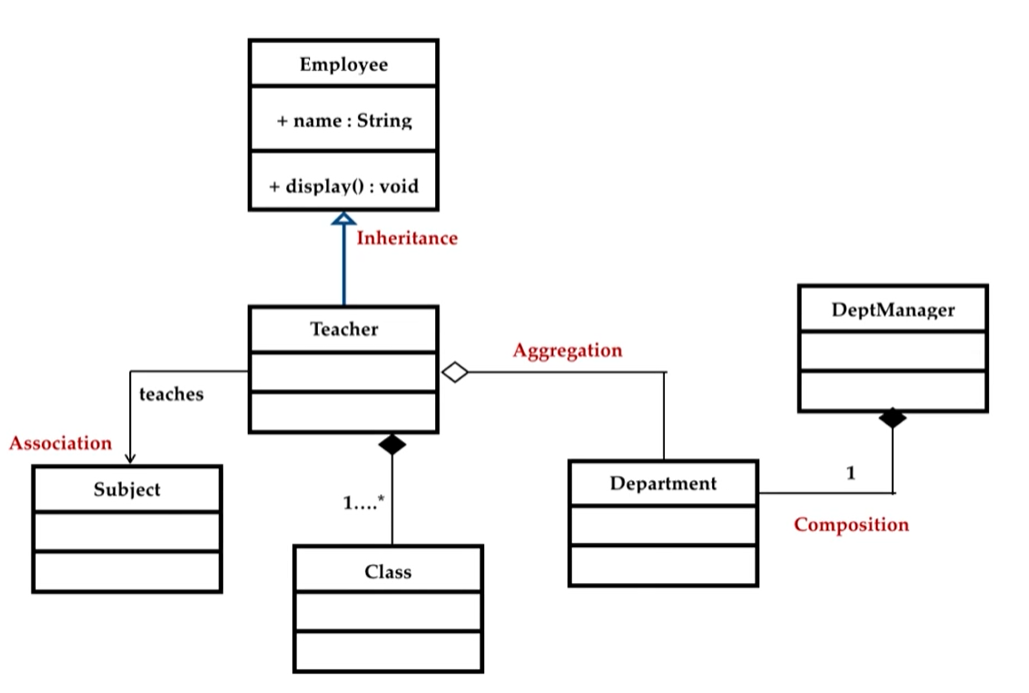
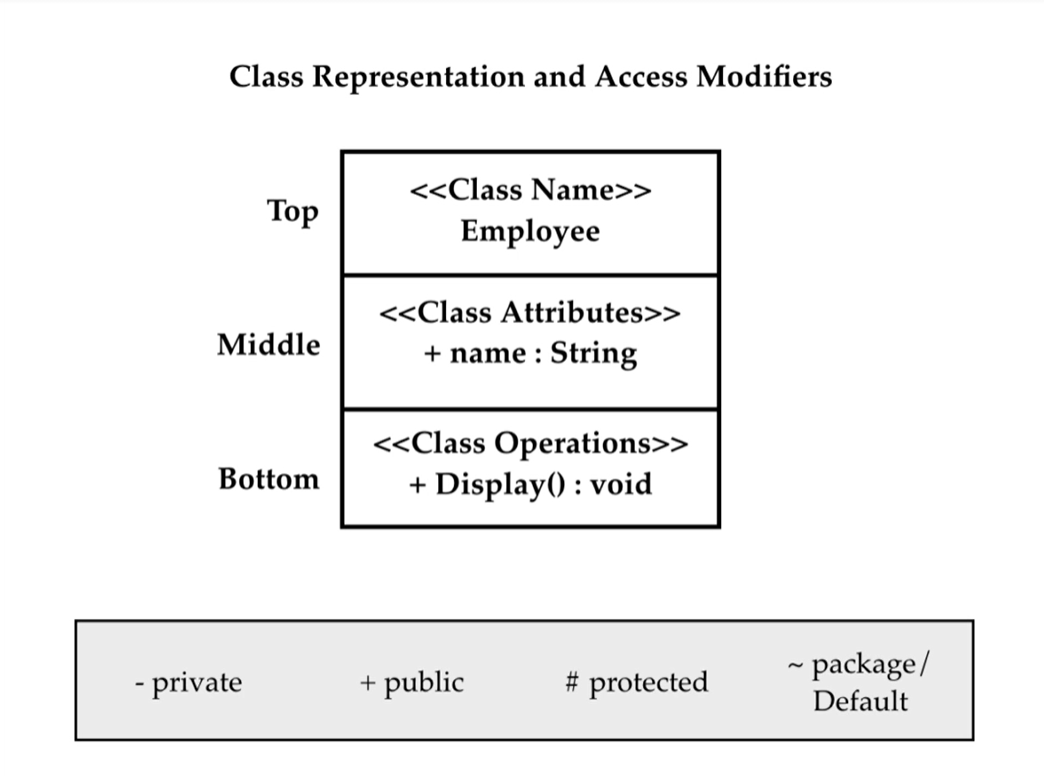
I**NTERFACE SEGREGATION PRINCIPLE**

-Parent classes shouldn’t enforce childs to implement unnecessarry functions, should be separated into different interfaces.

**DEPENDENCY INVERSION PRINCIPLE**

**-** Should be based on abstractions and not on concrete implementations

**UML**

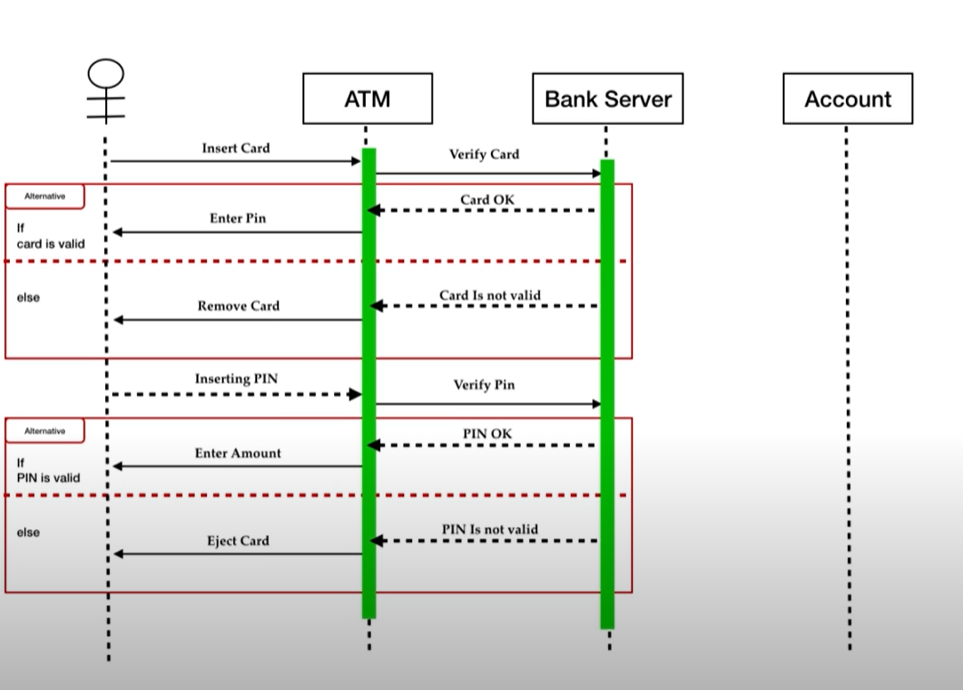


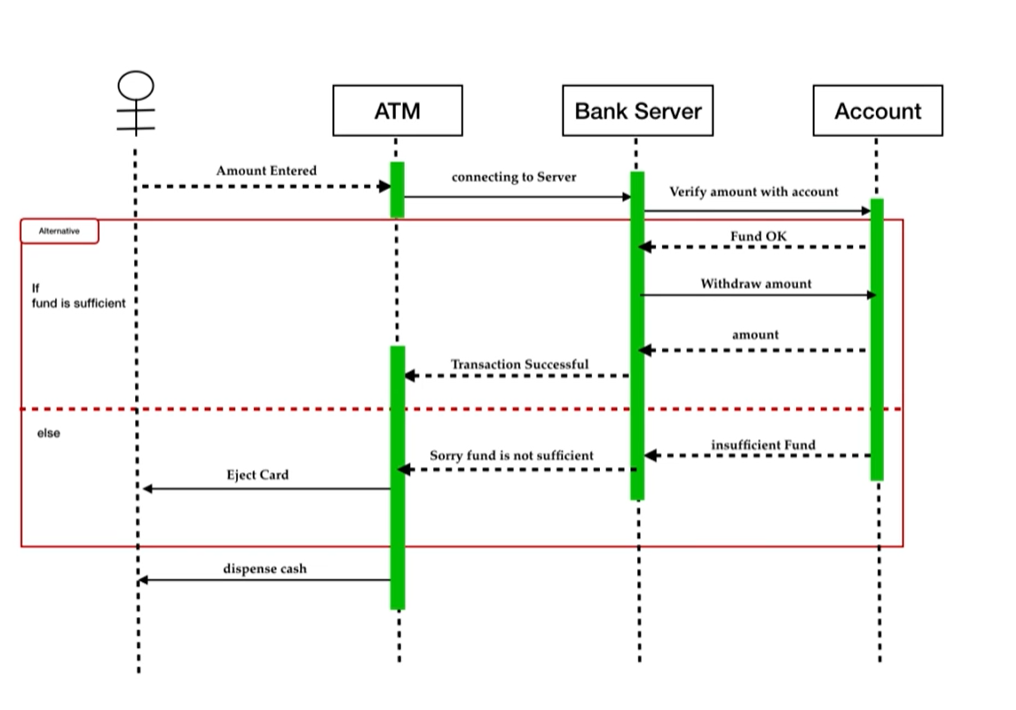
The numbers denote the number of classes we can have linked,

1 – Department can have one department manager

1.... - Class can have 1 or more teachers

Sequence Diagrams



**Design Patterns**

**Types:**

* **Structural Design Pattern**These patterns are more geared towards the ease of design by identifying a simple way to create relationship between entitites.

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Facade
6. Flyweight
7. Proxy

* **Behavioural Design Pattern**These Patterns are concerned with the interaction and responsibility of the objects such that they can talk to each other and still be loosely coupled.

1. Chain of Responsibility Patterns
2. Command Patterns
3. Interpretor Patterns
4. Iterator Patterns
5. Mediator Patterns
6. Memento Patterns
7. Observer Patterns
8. State Patterns
9. Strategy Patterns
10. Template Patterns
11. Visitor Patterns

* **Creational Design Pattern**

These patterns are concerned with the creation of objects or the complexity associated with it.

Examples:

1. Prototype Pattern
2. Builder Patterns
3. Singleton Pattern
4. Factory Method Patterns
5. Abstract Factory Pattern

List of derivatives are not same as list of bases.